ESP8266Sketch6_ServeurWeb1

Ce programme utilise un module **ESP8266** pour créer un serveur Web capable de contrôler une LED à distance via une connexion WiFi. L'utilisateur peut allumer ou éteindre la LED à l'aide de commandes envoyées depuis un navigateur web. La LED est contrôlée via une interface simple qui repose sur des requêtes HTTP.

```
#include <Arduino.h>
#include "ESP8266WiFi.h"

const byte led = LED_BUILTIN;
const char *SSID = "XXXXXX";
const char *PASSWORD = "123456";

WiFiServer server(80);
void setup()
{
  Serial.begin(115200);
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  WiFi.begin(SSID, PASSWORD);
  Serial.print("Connexion au réseau Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connecté à l'adresse IP :");
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop()
{
  WiFiClient client = server.accept();
  if (!client)
  {
    return;
  }
  while (!client.available())
  {

  }
  String request = client.readStringUntil('\r');
  Serial.println(request);
  client.flush();
  int value = LOW;
  if (request.indexOf("/LED=ON") != -1)
  {
    digitalWrite(led, HIGH);
    value = HIGH;
  }
  else if (request.indexOf("/LED=OFF") != -1)
  {
    digitalWrite(led, LOW);
    value = LOW;
  }
```

```
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html; charset=UTF-8");
    client.println("");
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.print("<h1>État de la LED: ");
    client.print(value == HIGH ? "Allumée" : "Éteinte");
    client.println("</h1>");
    client.println("<br><br>");
    client.println("Changez l'état de la LED : <br>");
    client.println("<a href=\"/LED=ON\">Allumer</a><br>");
    client.println("<a href=\"/LED=OFF\">Éteindre</a><br>");
    client.println("</html>");
    Serial.println("Réponse envoyée");
    delay(200);
  }
}
```

```
#include <Arduino.h>
#include "ESP8266WiFi.h"
```

- **<Arduino.h>** : Cette bibliothèque fournit les fonctionnalités de base pour programmer le module ESP8266, telles que la gestion des ports d'entrée/sortie et la communication série.
- **<ESP8266WiFi.h>** : Cette bibliothèque permet de gérer la connexion WiFi, la création d'un serveur Web, et la manipulation des connexions réseau sur l'ESP8266.

```
const byte led = LED_BUILTIN;
const char *SSID = "XXXXXXXX";
const char *PASSWORD = "123456";

WiFiServer server(80);
```

- **led** : Définit la broche où est connectée la LED. **LED_BUILTIN** est une constante prédéfinie qui se réfère à la LED intégrée à l'ESP8266.

- **SSID** et **PASSWORD** : Ce sont des constantes qui contiennent le nom du réseau WiFi (SSID) et le mot de passe pour permettre à l'ESP8266 de se connecter au réseau.

- **WiFiServer server(80)** : Initialise un serveur web sur le port **80**, le port standard pour le trafic HTTP.

```
void setup()
{
  Serial.begin(115200);
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);

  WiFi.begin(SSID, PASSWORD);
  Serial.print("Connexion au réseau Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
    Serial.println("\nWiFi connecté à l'adresse IP :");
    Serial.println(WiFi.localIP());
    server.begin();
  }
```

- **Serial.begin(115200)** : Démarre la communication série à une vitesse de 115200 bauds pour envoyer des messages de diagnostic au moniteur série.
- **pinMode(led, OUTPUT)** : Définit la broche où est connectée la LED comme une sortie.
- **digitalWrite(led, LOW)** : Initialise la LED en l'éteignant (niveau bas).
- **WiFi.begin(SSID, PASSWORD)** : Tente de connecter l'ESP8266 au réseau WiFi en utilisant le SSID et le mot de passe fournis.
- **while (WiFi.status() != WL_CONNECTED)** : Cette boucle attend que l'ESP8266 soit connecté au réseau WiFi, tout en affichant un point . dans le moniteur série pour chaque tentative.
- **Serial.println(WiFi.localIP())** : Affiche l'adresse IP locale assignée à l'ESP8266 une fois qu'il est connecté au réseau.
- **server.begin()** : Démarre le serveur web sur le port 80.

```
WiFiClient client = server.accept();
if (!client)
{
  return;
}
```

- **WiFiClient client = server.accept();** : Cette commande attend une connexion d'un client (généralement un navigateur web). Si un client se connecte au serveur, il est accepté et un objet **client** est créé pour gérer cette connexion.

- **if (!client)** : Si aucun client ne se connecte, la fonction quitte immédiatement grâce à ce test. Cela permet d'économiser des ressources lorsque le serveur est inactif.

```
while (!client.available())
{

  String request = client.readStringUntil('\r');
  Serial.println(request);
  client.flush();
```

- **while (!client.available())** : Cette boucle attend que le client envoie une requête complète. La fonction **client.available()** retourne vrai lorsqu'il y a des données à lire (la requête HTTP du client est prête à être traitée).

- **request = client.readStringUntil('\r');** : La requête HTTP envoyée par le client est lue ligne par ligne jusqu'à rencontrer un caractère de retour chariot (\r). La ligne lue est stockée dans la variable **request**.

- **Serial.println(request);** : Cette ligne affiche la requête dans le moniteur série pour un diagnostic en temps réel.

- **client.flush();** : Vide le buffer du client, supprimant toute donnée en attente non traitée pour garantir que seule la nouvelle requête soit analysée.

```
int value = LOW;
if (request.indexOf("/LED=ON") != -1)
{
  digitalWrite(led, HIGH);
  value = HIGH;
}
else if (request.indexOf("/LED=OFF") != -1)
{
  digitalWrite(led, LOW);
  value = LOW;
}
```

- **Contrôle de la LED** : Le serveur détecte si la requête HTTP contient **/LED=ON** ou **/LED=OFF**, ce qui correspond aux commandes pour allumer ou éteindre la LED. Cela

se fait grâce à la fonction **request.indexOf()**, qui vérifie si la chaîne correspondante est présente dans la requête.

- Si **/LED=ON** est détecté, la LED est allumée avec **digitalWrite(led, HIGH);**.
- Si **/LED=OFF** est détecté, la LED est éteinte avec **digitalWrite(led, LOW);**.

- **Mise à jour de la variable value** : La variable **value** est mise à jour pour refléter l'état actuel de la LED (soit **HIGH** pour allumée, soit **LOW** pour éteinte). Cela sera utilisé plus tard pour afficher l'état dans la réponse HTML.

```
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html; charset=UTF-8");
client.println("");
client.println("<!DOCTYPE HTML>");
client.println("<html>");
client.print("<h1>État de la LED: ");
client.print(value == HIGH ? "Allumée" : "Éteinte");
client.println("</h1>");
client.println("<br><br>");
client.println("Changez l'état de la LED : <br>");
client.println("<a href=\"/LED=ON\">Allumer</a><br>");
client.println("<a href=\"/LED=OFF\">Éteindre</a><br>");
client.println("</html>");
Serial.println("Réponse envoyée");
delay(200);
```

Réponse HTTP générée par le serveur :

- **En-têtes HTTP** : Le serveur commence par envoyer les en-têtes de réponse, notamment **HTTP/1.1 200 OK** pour indiquer que la requête a été traitée avec succès et **Content-Type: text/html; charset=UTF-8** pour signaler que le contenu renvoyé est une page HTML.

- **Contenu HTML** : La réponse contient une simple page HTML. Elle affiche l'état actuel de la LED dans un titre **<h1>**, soit "Allumée" soit "Éteinte", selon la valeur de **value**. Ensuite, deux liens sont générés :
  - Un lien pour allumer la LED (lien vers **/LED=ON**).
  - Un lien pour éteindre la LED (lien vers **/LED=OFF**).